REMARKS/ARGUMENTS

The specification has been amended on page 7 lines 7-16, and claims 60 and 61 have been amended, to say that the network file server is further programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface. This is shown in applicants' FIG. 3 in which the "UNCACHED MULTI-THREADED WRITE INTERFACE" 63 receives "SECTOR-ALIGNED WRITES" and the arrow from the uncached write interface 63 is labeled "INVALIDATE CACHE BLOCKS INCLUDING SECTORS BEING WRITTEN." This is further described in applicants' specification on page 13 lines 17-22 as follows:

> The uncached multi-threaded write interface 63 is used for sector-aligned writes to the file system 54. Sectors of data (e.g., 512 byte blocks) are read from the message buffers (53 in FIG. 2) and written directly to the cached disk array 29. For example, each file block is sector aligned and is 8 K bytes in length. When a sector-aligned write occurs, any cache blocks in the file system cache that include the sectors being written to are invalidated. In effect, the uncached multi-threaded write interface 63 commits file data when writing the file data to the file system 54 in storage.

See also FIGS. 5-6 steps 81 and 85-90 as described in applicants' specification on page 17 line 312 to page 18 line 14.

In paragraph 3 on page 2 of the Official Action, claims 1-4, 10, 11, 15, 22, 29, 32, 33-36, 44, 45, 49, 55, and 58-73 were rejected under 35 U.S.C. 102(b) as being anticipated by Xu et al. U.S. 6,324,581 B1. In response, claims 29 and 55 have been cancelled, and claims 60 and 61 have been amended.

"For a prior art reference to anticipate in terms of 35 U.S.C. § 102, every element of the claimed invention must be identically shown in a single reference." Diversitech Corp. v. Century Steps, Inc., 7 U.S.P.Q.2d 1315, 1317 (Fed. Cir. 1988), quoted in In re Bond, 910 F.2d 831,15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990) (vacating and remanding Board holding of anticipation; the elements must be arranged in the reference as in the claim under review, although this is not an *ipsis verbis* test).

Applicants' claim 1 calls for a network file server responding to a concurrent write request from a client for access to a file by a sequence of seven specific steps performed in a specific order. The network file server responds by:

(a) obtaining a lock for the file and then

(b) preallocating a metadata block for the file; and then

(c) releasing the lock for the file; and then

(d) asynchronously writing to the file; and then

(e) obtaining the lock for the file; and then

(f) committing the metadata block to the file; and then

(g) releasing the lock for the file.

The applicants' drawings show these steps (a) to (g) in FIG. 11 in boxes 101, 102, 103, 104-105, 1-6, 107, and 108, respectively, as described in applicants' specification on page 27 lines 3-23.

With respect to applicants' step (a) of obtaining a lock for the file, page 2 of the Official action cites Xu column 3, line 60 through column 4, line 12. However, this same passage is cited for on page 3 of the Official Action for applicants' step (e) of obtaining a lock for the file. With respect to applicants' step (b) of preallocating a metadata block for the file, page 2 of the Official Action cites Xu column 8, lines 36-64. With respect to applicants' step (c) of releasing the lock for the file, the Official Action cites Xu column 9, lines 21-39. However, this releasing of a lock for the file in Xu occurs after writing data to the file, and not before as recited in applicants' claim 1 steps (c) and (d), and also this same passage is cited on page 3 of the Official Action for applicants' step (g) of releasing the lock for the file. With respect to applicants' step (d) of asynchronously writing to the file, page 2 of the Official Action cites Xu column 9, lines 21-39. However, this writing occurs in Xu when the lock of Xu column 3, line 60 through column 4, line 12 is held by the data mover that accesses the data storage locations specified by the metadata of the file. With respect to applicants' step (e) of obtaining the lock for the file, page 3 of the Official Action again cites Xu column 3, line 60 though column 4, line 12, which occurs in Xu before the writing of Xu column 9, lines 21-29, instead of after, as recited in applicants' steps (d) and (e). With respect to applicants' step (f) of committing the metadata block to the file, page 3 of the Official Action cites Xu column 8, lines 65 through column 9, line 39. With respect to applicants' step (g) of releasing the lock for the file, page 3 of the Official Action again cites Xu column 9, lines 65 21-39.

It is respectfully submitted that Xu fails to disclose the applicants' step (c) of releasing the lock for the file before the step (d) of asynchronously writing to the file for Xu's network file server responding to a write request from a client. Xu also fails to disclose the applicants' step (e) of obtaining the lock for the file after the step (d) of asynchronously writing to the file. Nor are applicants' steps (c) and (e) inherent or obvious from Xu because Xu's lock of column 3, line 60 through column 4, line 12 is a lock granted by a first data mover computer to a second data mover so that the second data mover may access data storage locations of the file. The lock on the file as recited in applicants' claim 1 is not a lock for accessing data storage location of the file because it is released in applicants' step (c) prior to applicant's step (d) of asynchronously writing to the file, and again obtained in applicants' step (e) after applicant's step (d) of asynchronously writing to the file. Instead, the lock on the file as recited in applicants' claim 1 is associated with the step (b) of preallocating a metadata block for the file and is associated with the step (f) of committing the metadata block to the file, because the lock on the file as recited in applicants' claim 1 is obtained in step (a) before step (b) and then released in step (c) after step (b), and it is again obtained in step (e) before step (f) and then released in step (g).

Nor does Xu need or suggest such a lock on the file to be associated with the step (b) of preallocating a metadata block for the file and associated with the step (f) of committing the metadata block to the file in this fashion as claimed in claim 1 because Xu discloses an entirely different mechanism to control access to preallocating metadata blocks for the file and committing metadata blocks to the file. This entirely different mechanism uses file system configuration by assigning a data mover to each file so that this Owner data mover has exclusive

control over block allocation for the file and committing the metadata blocks to the file. See Xu col. 8, lines 47-54 and Xu col. 33 line 65 to col. 34 line 9. As disclosed in Xu col. 8 lines 36-54:

> In contrast to FIG. 1, the network file server architecture in FIG. 2 includes a data bypass path 48 between the first data mover 41 and the second file system 44 in order to bypass the second data mover 42, and a data bypass path 49 between the second data mover 42 and the first file system 43 in order to bypass the first data mover 41. It is possible for each of the data movers 41, 42 to access data in each of the file systems 43, 44, but if a data mover does not own the file access information for the file system to be accessed, then the data mover should ask the owner for permission to access the file system, or else a data consistency problem may arise. For example, when the first data mover 41 receives a file access request from its client 46, it accesses its directory of file ownership information to determine whether or not it owns the file system to be accessed. If the first data mover 41 does not own the file system to be accessed, then the first data mover 41 sends a metadata request to the data mover that owns the file system to be accessed. For example, if the first client 46 requests access to the second file system 44, then the first data mover 41 sends a metadata request to the second data mover 42.

As disclosed in Xu col. 33 line 56 to col. 34 line 9:

> There are several ways that the Owner can allocate disk blocks for the secondary data mover. In a preferred implementation, the secondary data mover tells the Owner that it wants to grow the file for some number of disk blocks. The Owner does the blocks allocation and allocates proper indirect blocks for the growth and informs the secondary data mover. Therefore, a secondary data mover

> works on the new file's metadata During this allocation, the blocks are not logged inside the log on the Owner and the file's in-memory inode structure is neither changed nor logged. When the secondary data mover sends the metadata back, the inode and indirect blocks are updated and logged. Some unused blocks are also reclaimed, because the free disk blocks are not shareable across different files inside one shadow file system. This makes ShFS's behavior different from that of UFS. Since SHFS does not have the usual file system structures, it does not support many of the normal file system operations, like name lookup. For those operations, ShFS can just return a proper error code as SFS currently does.

With respect to applicants' dependent claim 3, page 3 of the Official Action cites col. 8 line 65 through col. 9 line 39 for a teaching of copying data from an original indirect block for the file to the metadata block for the file, the original indirect block of the file having been shared between the file and a read-only version of the file. However, it is not understood how this passage or any other passage in Xu discloses "the original indirect block of the file having been shared between the file and a read-only version of the file." For example, compare Xu to applicants' FIG. 16 and the written description in applicants' specification on page 15 lines 8-11.

With respect to dependent claim 4, page 3 of the Official Action cites Xu col. 11 lines 20-37 and column 13 lines 36-42.. However, these passages relate to shared data access among clients, and do not disclose details of write access to a metadata block for a file. As discussed above with respect to Xu col. 33 line 56 to col. 34 line 9, for each file, a particular data mover owner of the file is assigned by way of file system configuration to have exclusive control over write access to a metadata block for a file. If another client or data mover (called a secondary

data mover) is to perform a write operation on the file that would need to change the metadata block for a file, the secondary gets the metadata and a write lock on the file, but still the secondary sends the new metadata back to the Owner and the Owner updates and logs the inode and indirect block. It is not seen where Xu discloses that writing to such an inode or indirect block would occur concurrently for more than one client instead of serially by the Owner having exclusive access to the metadata blocks of the file.

With respect to applicants' dependent claim 11, page 3 of the Official Action again cites Xu col. 8 line 65 through column 9, line 39. However, these passages fail to disclose gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file. Instead, as discussed above with respect to applicants' claim 4, as disclosed in Xu col. 33 line 56 to col. 34 line 9, for each file, a particular data mover owner of the file is assigned by way of file system configuration to have exclusive control over write access to a metadata block for a file. If another client or data mover (called a secondary data mover) is to perform a write operation on the file that would need to change the metadata block for a file, the secondary gets the metadata and a write lock on the file, but still the secondary sends the new metadata back to the Owner and the Owner updates and logs the inode and indirect block. It is not seen where Xu discloses that such a metadata block update sequence would include gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the

preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file. In addition, if the Owner would gather together preallocated metadata blocks for a plurality of client write requests to the file, it would not need to obtain a lock on the file to commit the gathered preallocated metadata blocks because the Owner has been configured to have exclusive access to the metadata of the file.

With respect to applicants' independent claim 15, as discussed above with reference to applicants' claim 11, it is respectfully submitted that Xu fails to disclose gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

With respect to applicants' independent claim 32, as discussed above with reference to applicants' claim 1, it is respectfully submitted that Xu fails to disclose applicants' step c) of releasing the allocation mutex of the file [prior to the step d) of issuing asynchronous write requests for writing to the file], and applicants' step f) of obtaining the allocation mutex for the file [after the step d) of issuing asynchronous write requests for writing to the file]. In addition, the lock of Xu column 3, line 60 through col. 4, line 12 appears to be a read lock or a write lock, and not an allocation mutex. Moreover, as discussed above with reference to Xu col. 8, lines 47-

54 and Xu col. 33 line 65 to col. 34 line 9, a data mover Owner of a file does not need an allocation mutex because it has been configured to have exclusive ownership and access to the metadata blocks of the file.

With respect to dependent claim 33, applicants respectfully traverse for the reasons given above with reference to claim 1.

With respect to dependent claim 35, applicants respectfully traverse for the reasons given above with reference to claim 3.

With respect to dependent claim 36, applicants respectfully traverse for the reasons given above with reference to claim 4.

With respect to dependent claim 45, applicants respectfully traverse for the reasons given above with reference to claim 11.

With respect to independent claim 49, applicants respectfully traverse for the reasons given above with reference to claim 15.

With respect to independent claim 58, applicants respectfully traverse for the reasons given above with reference to claim 32.

With respect to dependent claim 59, claim 59 has been amended to more clearly distinguish Xu by reciting that the network file server is further programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface. It is not seen where Xu discloses that a network file server that is programmed to invalidate cache blocks in the file system cache including sectors being written to by an uncached write interface.

With respect to independent claim 61, claim 61 has been amended to more clearly distinguish Xu by reciting that the network file server is further programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface. It is not seen where Xu discloses that a network file server that is programmed to invalidate cache blocks in the file system cache including sectors being written to by an uncached write interface.

In paragraph 5 on page 8 of the Official Action, claims 5-9, 12-14, 16-21, 23-28, 30, 31, 37-43, 46-48, 50-54, 56, and 57 were rejected under 35 U.S.C. 103(b) as being unpatentable over Xu in view of Marcotte U.S. 6,449,614. In response, claims 30-31 and 56-57 have been cancelled. Otherwise, applicants respectfully traverse and respectfully submit that the other rejected claims would not have been obvious from Xu in view of Marcotte.

Marcotte discloses an interface system and methods for asynchronously updating a share resource with locking facility. Tasks make updates requested by calling tasks to a shared resource serially in a first come first served manner, atomically, but not necessarily synchronously, such that a current task holding an exclusive lock on the shared resource makes the updates on behalf of one or more calling tasks queued on the lock. Updates waiting in a queue on the lock to the shared resource may be made while the lock is held, and others deferred for post processing after the lock is released. Some update requests may also, at the calling application's option, be executed synchronously. Provision is made for nested asynchronous locking. Data structures (wait_elements) describing update requests may queued in a wait queue for update requests awaiting execution by a current task, other than the calling task, currently

holding an exclusive lock on the shared resource. Other queues are provided for queuing data structures removed from the wait queue but not yet processed; data structures for requests to unlock or downgrade a lock; data structures for requests which have been processed and need to be returned to free storage; and data structures for requests that need to be awakened or that describe post processing routines that are to be run while the lock is not held. (Abstract.)

With respect to applicants' dependent claims 5-9, 12-14, 16-21, 23-24, 37-43, and 46, these claims depend from the independent claims 1, 15, 33, and 49. Xu has been distinguished above with respect to the independent claims 1, 15, 33, and 49, and Marcotte does not provide the limitations of these independent claims that are missing from Xu. Therefore the dependent claims 5-9, 12-14, 16-21, 23-24, 37-43, and 46 are patentable over the proposed combination of Xu and Marcotte.

Applicants' dependent claim 5 adds to claim 1 the limitations of "wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing the partial write to the new block." Applicants' partial block conflict queue 73 is shown in applicants' FIG. 4 and described in applicant's specification on page 15 lines 17-22 as follows:

> The preallocation method allows concurrent writes to indirect blocks within the same file. Multiple writers can write to the same indirect block tree

> concurrently without improper replication of the indirect blocks. Two different indirect blocks will not be allocated for replicating the same indirect block. The write threads use the partial block conflict queue 73 and the partial write wait queue 74 to avoid conflict during partial block write operations, as further described below with reference to FIG. 13.

See also applicants' FIG. 13 and specification page 28 line 22 to page 29 line 7; and FIG. 17 and page 34 line 7 to page 35 line 5.

With respect to applicants' dependent claim 5, page 8 of the Official Action recognizes that Xu fails to explicitly recite the limitations added by dependent claim 5, and says that Marcotte teaches these limitations, citing column 13, lines 35 through col. 14, line 43. However, Marcotte column 13, lines 35 through col. 14, line 43 deals with managing an I/O device holding queue above a device for queuing pending I/O's if the number of I/O's issued to the device exceeds an a threshold that is adjustable by an application. It is not seen where Marcotte discloses a partial write to a new block that has been copied at least in part from an original block of the file. Nor is it seen where Marcotte discloses checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing the partial write to the new block.

"[R]ejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." In re Kahn, 441 F. 3d 977, 988 (Fed. Cir. 2006).

A fact finder should be aware of the distortion caused by hindsight bias and must be cautious of arguments reliant upon ex post reasoning. See KSR International Co. v. Teleflex Inc., 550 U.S. __, 82 USPQ2d 1385 (2007)), citing Graham, 383 U. S. at 36 (warning against a "temptation to read into the prior art the teachings of the invention in issue" and instructing courts to "guard against slipping into the use of hindsight.").

Applicants' claim 6 is similar to claim 5 in that it also adds to claim 1 the limitations of wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block. With respect to dependent claim 6, page 9 of the Official Action also recognizes that Xu fails to explicitly recite the limitations added by dependent claim 6, and says that Marcotte teaches these limitations, again citing column 13, lines 35 through col. 14, line 43. However, Marcotte column 13, lines 35 through col. 14, line 43 deals with managing an I/O device holding queue above a device for queuing pending I/O's if the number of I/O's issued to the device exceeds an a threshold that is adjustable by an application. It is not seen where Marcotte discloses a partial write to a new block that has been copied at least in part from an original block of the file. Nor is it seen where Marcotte discloses checking a partial block conflict queue for a conflict with a concurrent write to the new block.

With respect to applicants' dependent claim 12, applicants respectfully submit that Xu does not further teach checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file. As discussed above with reference to applicants' claim 1, in response to a concurrent

write request from a client, Xu does not obtain a lock for the file after the asynchronously writing to the file. Nor dos Marcotte column 12, line 7 through column 13, line 32 disclose these limitations missing from Xu or specifically deal with committing "metadata blocks" to a file.

Applicants' independent claim 13 recites "wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block, and then performing the partial write to the new block." Thus, Applicants' independent claim 13 is patentable over Xu in combination with Marcotte for the reasons given above with reference to applicants' claim 5.

Applicants' dependent claim 14 is dependent upon claim 13 and therefore is also patentable over Xu in combination with Marcotte for the reasons given above with reference to applicants' claim 13.

Applicants' dependent claim 16 adds to claim 15 the limitations of claim 12 and therefore is patentable over Xu in combination with Marcotte for the reasons given above with reference to applicants' claim 12.

Applicants' dependent claim 17 adds to claim 15 the limitations of "wherein the network file server includes disk storage containing a file system, and a file system cache storing data of blocks of the file, and the method further includes the network file server responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and

invalidating the specified blocks of the file in the file system cache." Page 14 of the Official Action recognizes that Xu fails to explicitly recite these limitations, and cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 18 adds to claim 17 the limitations of "the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale." Page 15 of the Official Action again cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 19 adds to claim 18 the limitations of "the network file server checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block from the file system in the disk storage, and then again checking whether the file block is in the file system cache." Page 15 of the Official Action again cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 20 adds to claim 18 the limitations of "the network file server setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block." Page 15 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 21 adds to claim 18 the limitations of "the network file server maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block." Page 16 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

With reference to applicants' independent claim 25, pages 17-18 of the Official Action recognize that Xu fails to explicitly recite the network file server responding to concurrent write requests by writing new data for specified blocks of the file to disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache. (See, e.g., applicants' FIG. 10, steps 515

and 516; applicants' spec., page 23 lines 19-23 and page 24 lines 7-15.) Page 18 of the Official Action further recognizes that Xu fails to explicitly recite the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become state, and not writing to the file system cache a file block that has become stale. (See, e.g., applicants' FIG. 10, steps 92, 97, 513, 98; applicants' spec., page 23 lines 5-17; page 24 lines 16-22 .) Page 19 of the Official Action cites col. 12 line 7 through column 12, line 32 for all of these limitations not explicitly recited in Xu. However, it is not understood how all of the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Marcotte column 12 line 7 through column 13, line 32, deals generally with maintaining a list of lock waiters. As shown in Marcotte FIG. 9, when a task waits for a lock, it queues its WAIT_ELEMENT to the lock using lock or wait routine 175 and also adds it WAIT_ELEMENT to a global list or queue 230 before it waits, and remotes it from the global list 230 after it waits. As shown in Marcotte FIG. 11, do_wait 240 is executed each time a thread needs to go into a wait for a lock . Step 241 executes a lock_UpdateResource procedure. Step 242 waits on ecb; and upon receiving it, step 243 executes the lock_Update Resource procedure. Marcotte says that in this way, the task waiting on a lock 100 will only actually suspend itself on the WAIT call. Adding and removing tasks (WAIT ELEMENTS) from global list 230 is done in a manner guaranteed to make the calling task wait. FIG. 12 shows Referring to FIG. 12, add_to_global routine 245 (with waitp-arg) includes step 246 which determines if prevent flag is null; if so, step

248 posts an error code in the ecb field 108 of the WAIT_ELEMENT being processed; and, if not, step 247 adds the WAIT_ELEMENT (in this case, task 232) to the head of global list 230. FIG. 13 shows a remove_from_global routine 250 (with waitp=arg) including step 251 which determines if prevent flag 124 is null. If so, return code (rc) is set to zero; and if not, this WAIT_ELEMENT (say, 233) is removed from global list 230. In step 254, the return code (rc) is returned to the caller. FIG. 14 shows a return_wait routine 260 (with waitp=prc) including step 261 which determines if waitp is null. If not, the WAIT_ELEMENT pointed to by waitp is returned to free storage. Return wait 260 is the post processing routine for remove_from_global 250, and remove_from_global communicates the address of the WAIT_ELEMENT via its return code, that is input to return_wait (automatically by the resource update facility) as prc. Return_wait 260 returns the WAIT_ELEMENT to free storage. Since routine 260 is a post processing routine, the free storage return is NOT performed while holding the lock. This shows the benefits of a post processing routine, and passing the return value from a resource update routine to the post processing routine. FIG. 15 shows quiesce_global 265 wakes up all waiters and in step 267 tells them that the program is terminating due to error by way of prevent flag 124 being set to 1 in step 266. In step 268 pointer 231 and 234 are cleared so global list 230 is empty.

Thus, it is not understood how Marcotte's general teaching of a way of maintaining a list of lock waiters discloses the applicants' specific limitations of the network file server responding to concurrent write requests by writing new data for specified blocks of the file to disk storage without writing the new data for the specified blocks of the file to the file system cache. invalidating the specified blocks of the file in the file system cache, and responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file

system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become state, and not writing to the file system cache a file block that has become stale.

"[R]ejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." In re Kahn, 441 F. 3d 977, 988 (Fed. Cir. 2006). A fact finder should be aware of the distortion caused by hindsight bias and must be cautious of arguments reliant upon ex post reasoning. See KSR International Co. v. Teleflex Inc., 550 U.S. __, 82 USPQ2d 1385 (2007)), citing Graham, 383 U. S. at 36 (warning against a "temptation to read into the prior art the teachings of the invention in issue" and instructing courts to "guard against slipping into the use of hindsight.").

Applicant's dependent claim 26 adds to claim 25 limitations Applicants' dependent claim 19 adds to claim 18 the limitations of "the network file server checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block from the file system in the disk storage, and then again checking whether the file block is in the file system cache." Page 19 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how these specific limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 27 adds to claim 25 the limitations of "the network file server setting a read-in-progress flag for a file block upon finding that the file block is not in the

file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block." Page 19 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how these specific limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 28 adds to claim 25 the limitations of "the network file server maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block." Page 19 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how these specific limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

With respect to dependent claim 37, applicants respectfully traverse for the reasons given above with reference to claim 5.

With respect to dependent claim 38, applicants respectfully traverse for the reasons given above with reference to claim 6.

With respect to dependent claim 46, applicants respectfully traverse for the reasons given above with reference to claim 12.

With respect to independent claim 47, applicants respectfully traverse for the reasons given above with reference to claim 13.

With respect to dependent claim 48, applicants respectfully traverse for the reasons given above with reference to claim 14.

In view of the above, it is respectfully submitted that the application is in condition for allowance. Reconsideration and early allowance are earnestly solicited.

Respectfully submitted,

/ Richard C. Auchterlonie /

Richard C. Auchterlonie, Reg. No. 30,607

NOVAK DRUCE & QUIGG, LLP
1000 Louisiana, 53rd Floor
Houston, TX 77002
713-571-3460